Simulating Competing Risks Data

Nirupama Tamvada

Generating competing risks data through cause-specific hazards

We generate competing risks data based on cause-specific hazards as specified in Beyersmann et al. (2009).

- Each of the cause-specific hazards, $h_{0i}(t)$, is specified independently of the rest of the competing events
- Survival time T is simulated with all-cause hazard $h_{01}(t) + h_{02}(t) + ..+$
- Observed time $T = min(T_1, T_2, \tau)$
- For a given simulated survival time T, run a binomial experiment for which the probability that the individual fails from cause 1: $\frac{h_{01}(t)}{(h_{01}(t)+h_{02}(t)+..+)}$
- Generate censoring times by specifying a censoring distribution
- The `survsim` package that the simulating function is based on uses the Accelerated Failure Time (AFT) parameterization. To recover the coefficients from the Proportional Hazards (PH) parameterization for a Weibull distribution, we use the following relationship:

  $\beta_{\text{PH}} = -\beta_{\text{AFT}}.p$ where p represents the shape parameter of the Weibull distribution.

Steps to simulate competing risks data:

1. Survival time is first simulated by assuming an appropriate cause-specific distribution. For example, for the Weibull distribution, the following parameterization can be used to generate the survival times

$$T = \left(\frac{-log(u)}{\lambda \exp(\beta' x)} \right)^{\frac{1}{p}}$$

where $u \sim U(0,1)$

Similar to above, the exponential distribution can also be used as per the parameterization of Bender et al. (2005). This is the step performed in the `sim_surv_time` function

```
## simulate survival time according to Bender et al. (2005)
sim_surv_time <- function(row, betas, dist, lambda, gamma) {
  U <- stats::runif(1, min=0, max=1)
  eff <- sum(row * betas)

  if (dist=="weibull") {
    surv_time <- (-(log(U)/(lambda*exp(eff))))^(1/gamma)
  } else if (dist=="exponential") {
    surv_time <- -(log(U)/(lambda*exp(eff)))
  }
  return(surv_time)
}
```

2. Simulating high-dimensional co-variates with a block structure: We wish to simulate the covariates X in a high-dimensional setting from a multivariate normal distribution where variables are correlated pairwise with a constant correlation, as well as potentially with a block-wise correlation structure.

   We first consider the case of pairwise correlations that are constant as well as an $AR(1)$ covariance matrix.

```
# Trevor's function to generate multivariate normal with constant pairwise correlation
gen.x = function(n,p,rho){
 if(abs(rho)<1){
  beta=sqrt(rho/(1-rho))
 x0=matrix(rnorm(n*p),ncol=p) # x0 dim is n*p, standard normal (mean = 0, sd = 1) values
 z=rnorm(n)
 x=beta*matrix(z,nrow=n,ncol=p,byrow=F)+x0
 }
 if(abs(rho)==1){ x=matrix(z,nrow=n,ncol=p,byrow=F)}
return(x)
}

# Simulating constant correlation of 0.5
sim.x <- gen.x(250, 50, 0.5)

cor(sim.x)
```

```r
# Try AR1 correlation
covAR1 <- function(p, rho) {
    stopifnot(p >= 2 && rho >= 0)
    # https://statisticaloddsandends.wordpress.com/2020/02/07/generating-correlation-matri
    exponent <- abs(matrix(1:p - 1, nrow = p, ncol = p, byrow = TRUE) -
                        (1:p - 1))

    return(rho^exponent)
}

gen.x_AR1 = function(n,p,rho){
 if(abs(rho)<1){
  beta=sqrt(rho/(1-rho))
 x0=matrix(rnorm(n*p),ncol=p) # x0 dim is n*p, standard normal (mean = 0, sd = 1) values
 z=rnorm(n)
 x= mvrnorm(250, mu = rep(0, 50), Sigma = covAR1(p, rho)) + x0
 }
 if(abs(rho)==1){ x=matrix(z,nrow=n,ncol=p,byrow=F)}
return(x)
}

sim.data1 <- gen.x_AR1(250, 50, 0.5)
cor(sim.data1)
```

```r
library(matrixcalc)
library(Matrix)

## Simulate cause-specific survival time with respective event to simulate competing risks
# NOTE: the beta coefficients of the proportional hazards
#       modification can be recovered by: b_ph = -b_aft * gamma
#       See Morina (2017) p. 5716
sim_crisk_time <- function(x, outcome_betas, gamma, lambda, max_t) {

  # has to be a unnamed numeric vector
  eff <- as.numeric(x)

  # to store some stuff later
  a.ev <- vector()
  b.ev <- vector()
  pro <- vector()
  cshaz <- list()
```

```r
cause <- NA
nsit <- length(outcome_betas[[1]])

## Simulates the all cause hazard
suma <- vector()
for (m2 in seq_len(nsit)) {
  suma[m2] <- 0
  for (m1 in
       ) {
    suma[m2] <- suma[m2] + outcome_betas[[m1]][m2]*eff[m1]
  }
}

# Cause-specific hazard function, based on weibull distribution
# equation can be found in Morina & Navarro (2017, p. 5714)
for (k in seq_len(nsit)) {

  a.ev[k] <- lambda[k] + suma[k]
  b.ev[k] <- gamma[k]
  cshaz[[k]] <- function(t, r) {
    par1 <- eval(parse(text="a.ev[r]"))
    par2 <- eval(parse(text="b.ev[r]"))
    return((((1/par2)/exp(par1))^(1/par2))*t^((1/par2)-1))
  }
}

# Cumulative all-cause hazard function A
A <- function(t, y) {
  res <- 0
  for (k in seq_len(length(cshaz))) {
    res <- res + stats::integrate(cshaz[[k]], lower=0.001, upper=t, r=k,
                                  subdivisions=1000)$value
  }
  res <- res + y
  return(res[1])
}

# There was a while loop here in the original survsim code, which
# resampled values for u whenever the condition in the following if
# statement was not true. By resampling these cases that should be censored
# (because their survival time is greater than the maximal follow up time)
```

```
  # were replaced with artificially small survival times. This lead to wrong
  # observed coefficients in the sample for small max_t.

  # Instead of doing that, we simply denote any observation that fails
  # this test as censored. This works much better.
  u <- stats::runif(1)
  if (A(0.001, log(1-u))*A(max_t, log(1-u)) > 0) {
    return(c(time=max_t, cause=0))
  }

  # Numeric inversion method
  # max_t needs to be included. It defines the interval in which the root
  # will be searched for (See Beyersmann 2012; p. 49). Not a problem
  # anymore due to the changes above.
  tb <- stats::uniroot(A, c(0, max_t), tol=0.0001, y=log(1-u))$root

  # calculate probabilities for observing each cause, defined
  # by equation 2 in Beyersmann et al. (2009)
  sumprob <- 0
  for (k in seq_len(length(cshaz))) {
    sumprob <- sumprob + cshaz[[k]](tb, k)
  }

  for (k in seq_len(length(cshaz))) {
    pro[k] <- cshaz[[k]](tb, k) / sumprob
  }

  # draw observed cause using multinomial distribution
  cause1 <- stats::rmultinom(1, 1, prob=pro)
  for (k in seq_len(length(cshaz))) {
    if (cause1[k] == 1) {
      cause <- k
    }
  }

  # need to return both the time and the cause
  return(c(time=tb, cause=cause))
}
```

```
## function to simulate confounded competing risks data
#' @export
```

```r
sim_confounded_crisk <- function(n=500, lcovars=NULL, outcome_betas=NULL,
                                 group_beta=c(1, 0), gamma=c(1.8, 1.8),
                                 lambda=c(2, 2), treatment_betas=NULL,
                                 intercept=-0.5, gtol=0.001,
                                 cens_fun=function(n){stats::rweibull(n, 1, 2)},
                                 cens_args=list(), max_t=1.7) {

  check_inputs_sim_crisk_fun(n=n, lcovars=lcovars, outcome_betas=outcome_betas,
                             gamma=gamma, lambda=lambda,
                             treatment_betas=treatment_betas,
                             group_beta=group_beta, intercept=intercept,
                             gtol=gtol, cens_fun=cens_fun, cens_args=cens_args,
                             max_t=max_t)

  # set defaults to parameters used in authors simulation study
  if (is.null(lcovars)) {
    lcovars <- list(x1=c("rbinom", 1, 0.5),
                    x2=c("rbinom", 1, 0.5),
                    x3=c("rbinom", 1, 0.5),
                    x4=c("rnorm", 0, 1),
                    x5=c("rnorm", 0, 1),
                    x6=c("rnorm", 0, 1))
  }
  if (is.null(outcome_betas)) {
    outcome_betas <- list(c(log(1.8), log(1.8)/3),
                          c(log(1.3), log(1.3)/3),
                          c(0, 0),
                          c(log(1.8), log(1.8)/3),
                          c(log(1.3), log(1.3)/3),
                          c(0, 0))
  }
  if (is.null(treatment_betas)) {
    treatment_betas <- c(x1=0, x2=log(2), x3=log(1.5),
                         x4=0, x5=log(1.5), x6=log(2))
  }

  # draw i.i.d. random variables specified in lcovars
  covars <- data.frame(id=1:n)
  for (col in names(lcovars)) {
    if (lcovars[[col]][1]=="rnorm") {
      covars[, col] <- stats::rnorm(n, as.numeric(lcovars[[col]][2]),
```

```
                                  as.numeric(lcovars[[col]][3]))
  } else if (lcovars[[col]][1]=="runif") {
    covars[, col] <- stats::runif(n, as.numeric(lcovars[[col]][2]),
                                  as.numeric(lcovars[[col]][3]))
  } else if (lcovars[[col]][1]=="rbinom") {
    covars[, col] <- stats::rbinom(n, as.numeric(lcovars[[col]][2]),
                                   as.numeric(lcovars[[col]][3]))
  }
}
covars$id <- NULL

# assign binary treatment using logistic regression
if (length(treatment_betas)==1) {
  group_p <- intercept + (treatment_betas * covars[, names(treatment_betas)])
} else {
  group_p <- intercept + rowSums(treatment_betas *
                                   covars[, names(treatment_betas)])
}
group_p <- 1/(1 + exp(-group_p))

# in order to keep the positivity assumption,
# values of 0 and 1 can not be tolerated
group_p[group_p < gtol] <- gtol
group_p[group_p > (1 - gtol)] <- 1 - gtol

covars$group <- stats::rbinom(n=n, size=1, prob=group_p)

# add group_beta to outcome_betas
outcome_betas[[length(outcome_betas) + 1]] <- group_beta


# generate cause-specific survival times and cause
surv <- apply(covars, 1, sim_crisk_time, outcome_betas=outcome_betas,
              gamma=gamma, lambda=lambda, max_t=max_t)
surv <- as.data.frame(t(surv))

covars$time <- surv$time
covars$event <- surv$cause

# introduce random censoring if specified
if (!is.null(cens_fun)) {
```

```
    cens_time <- do.call(cens_fun, args=c(n=n, cens_args))

    covars <- within(covars, {
      event <- ifelse(time < cens_time, event, 0)
      time <- ifelse(time < cens_time, time, cens_time)
    })
  }

  return(covars)
}
```